

# Beyond the Benchmark Shape: Evaluating AI-Generated CUDA and Triton Kernels Under Realistic AI Workloads

---

*An evidence-grounded research perspective and preregisterable protocol for numerical correctness, distribution-shift robustness, hardware efficiency, and deployment-level value*

**Research article | 19 October 2026**

**Author:** [Author name], [verified professional role]

**Affiliation:** Deaimer (subject to organizational approval)

**Article type:** Research perspective and experimental proposal; no original experimental results are reported.

## Abstract

Large language models and coding agents can increasingly write GPU kernels, but a kernel that compiles, agrees with a reference on familiar inputs, and wins a timing test has not necessarily become a safe or economically meaningful replacement in an AI system. Performance depends on tensor shape, numerical format, memory layout, compiler behavior, GPU generation, launch mechanism, and the surrounding training or inference workload. Recent work has made important progress: KernelBench established executable kernel-generation evaluation; KernelBench-X documented category-dependent correctness and hardware-efficiency gaps; SOL-ExecBench introduced hardware-oriented performance bounds; Atrex-Bench incorporated trace-derived workload weights; and FastKernels studied production-oriented integration [6–10]. These advances make a further question testable: **how stable is an AI-generated kernel’s advantage when its evaluation distribution changes, and how much of that advantage survives deployment?**

This paper synthesizes the relevant CUDA software and hardware context, analyzes the complementary strengths of current benchmarks, and proposes the **Workload-Shift and Transfer Protocol (WSTP)**. WSTP combines held-out workload distributions, shape- and precision-aware correctness testing, asynchronous-safe timing, hardware-grounded profiling, anti-shortcut checks, and a staged path from isolated kernels to serving-level outcomes. It specifies falsifiable hypotheses, candidate and baseline controls, uncertainty reporting, a reproducibility manifest, and bounded claims about energy and cost. The proposal is intended for ML-systems researchers, GPU programmers, AI-agent evaluation teams, and infrastructure decision-makers. **It reports no new measurements, rankings, speedups, or deployed implementations.** Its proposed contribution is a transparent study design that can be implemented, challenged, and refined through independent experiments.

**Keywords:** NVIDIA CUDA; Triton; AI-generated GPU kernels; LLM inference; ML systems; workload shift; reproducible benchmarking; agent evaluation; numerical correctness; GPU infrastructure

## 1. Introduction: a fast kernel is not yet a useful kernel

Modern AI applications depend on highly tuned GPU kernels. Matrix multiplication, attention, normalization, reductions, data movement, and collective communication must be mapped onto devices whose computing throughput and memory behavior differ sharply from CPUs. CUDA provides a major programming and runtime substrate for NVIDIA GPUs, while frameworks and systems including PyTorch, Triton, CUTLASS, and inference engines expose different levels of abstraction to developers [1–4]. Progress at the kernel level can be consequential: a cheaper attention operation or more efficient data layout may influence model capacity, inference throughput, or latency. Yet **local speed and system value are not interchangeable.**

Consider a plausible, illustrative situation. An AI coding agent rewrites an RMS normalization kernel and reports a 1.4-times microbenchmark speedup on a fixed, contiguous tensor. The same implementation later receives a different stride layout, a very small batch, a longer sequence, or BF16 inputs containing values close to the reference tolerance boundary. It might remain correct, become

slower, exhibit excess memory traffic, or fail the numerical contract. If the optimized operator accounts for only a small fraction of serving time, even a genuine isolated speedup may scarcely change user-visible latency. These are hypotheses to investigate in a particular system, not observations from a completed experiment.

The research problem has changed as agents become able to produce large numbers of candidate kernels. Development cost shifts from writing every implementation manually to ensuring that each generated candidate obeys its semantics and is evaluated without shortcuts. Tests need to discriminate between a kernel that truly implements the computation and one that delegates to a library, specializes narrowly to visible shapes, uses permissive tolerances, or succeeds because timing accidentally measures asynchronous launch overhead. Some of these risks are documented by contemporary benchmark studies [7–11].

The purpose of this paper is not to suggest that existing researchers have ignored production realism. On the contrary, FlashInfer-Bench connects kernel generation with inference-system substitution; SOL-ExecBench supplies hardware-grounded goals; Atrex-Bench uses real serving traces; and FastKernels tests broader integration [8–10, 12]. The narrower opportunity is to make **distribution shift and deployment transfer explicit experimental outcomes**, with a predeclared decision rule instead of discovering only after the fact which shapes or performance metrics support a favorable claim.

The central thesis is therefore conditional: **an AI-generated kernel should be called deployment-ready only for an explicitly defined workload distribution, correctness contract, hardware/software environment, and acceptance threshold**. This framing is especially relevant to enterprise procurement, where an attractive single-number speedup can conceal operational risks, and to academic research, where an unrepeatably result cannot easily be built upon.

The paper makes three *proposal-level* contributions: it identifies a measurable gap between static benchmark success and shift-aware deployment evidence; it defines the Workload-Shift and Transfer Protocol (WSTP) with preregisterable hypotheses and controlled comparisons; and it specifies an artifact and reporting contract that

can support independent replication. Neither WSTP's effectiveness nor any comparative advantage is claimed as an established empirical result.

## 2. CUDA and the AI systems stack in 2026

### 2.1 From GPU threads to end-to-end workloads

In conventional CUDA C++, the host launches kernels to the device; kernels run as grids of thread blocks, whose threads execute in warps across streaming multiprocessors. Registers, shared memory, caches, and device memory give programmers several ways to trade storage capacity against data reuse. A performance optimization may change tiling, occupancy, memory coalescing, synchronization, or instruction mix [1]. Those choices matter because high arithmetic throughput is only useful if the operands arrive in time and the application has enough parallel work to keep the device occupied.

A useful first-order model is the roofline relation

$$\text{attainable FLOP/s} \leq \min(\text{peak compute FLOP/s}, \text{arithmetic intensity} \times \text{achievable memory bandwidth})$$
. Arithmetic intensity is the number of operations per byte of relevant memory traffic. The model offers a diagnosis, not a guarantee: register pressure, kernel-launch overhead, dependency stalls, cache behavior, and scheduling can all make a measured kernel slower than the bound suggests. Nsight Compute includes roofline-oriented profiling; Nsight Systems supplies the time-ordered context in which GPU kernels, CUDA API calls, memory transfers, and synchronization occur [4–5].

CUDA is an ecosystem rather than a single alternative to a deep-learning framework. PyTorch operations often dispatch to vendor libraries or compiled kernels. CUTLASS provides reusable abstractions for high-performance matrix and related computations, including architecture-dependent mixed-precision support. Triton enables tile-oriented GPU-kernel programming with compiler-managed lowering. NCCL supports GPU-aware communication collectives for multi-GPU applications. At the system level, inference engines coordinate batching, cached attention state, scheduling, and memory allocation [2–3, 13]. Each layer may hide or expose bottlenecks of the layers beneath it.

## 2.2 CUDA 13.4, tile programming, and precision

At the time of writing, NVIDIA’s archive lists CUDA Toolkit **13.4.3 (October 2026)** as the latest toolkit entry [14]. Version numbers are important because a benchmark is only reproducible if the driver, toolkit, compiler, library builds, architecture, and device configuration are recorded. CUDA 13.4 release notes discuss multi-process service changes and developer-preview support for the Vera Rubin platform. NVIDIA explicitly states that this Rubin preview is **not intended for benchmarking, performance analysis, or production deployment** [15]. Proposed performance experiments in this paper therefore must use supported, benchmarkable configurations, not preview support described as if it were deployment evidence.

The kernel-authoring landscape now includes multiple abstraction levels. CUDA C++ can expose thread-level details. Triton allows developers and agents to express blocked tensor operations. NVIDIA’s cuTile Python provides tile-level programming and has a documented September 2026 release with CUDA Toolkit 13.4-related features [16]. CUTLASS documents FP16/BF16, FP8, and, where supported, block-scaled formats such as NVFP4 and MXFP variants [2]. These options may reduce programming effort or access specialized hardware, but they do not eliminate requirements for shape correctness, numerical validation, and hardware-specific profiling.

Precision is central to the evaluation contract. An FP8 or FP4 computation is not automatically required to produce the same bit pattern as an FP32 reference, and different algorithms can use different accumulation orders. A scientifically honest comparison must state the **intended numerical semantics** in advance: accepted formats, accumulation policy, permitted error, treatment of NaN and infinity, and any downstream quality target. Where a format or matrix instruction is unavailable on one GPU, the study must mark the test unsupported instead of silently replacing it with a different computation.

## 2.3 Why training and inference demand different evidence

Training mixes forward computation, backpropagation, activation storage, optimizer updates, data ingestion, and often collective communication. Kernel optimizations can change numerical trajectories, so step time alone cannot establish improved *training*

*quality or time to a fixed validation target.* NCCL handles topology-aware primitives such as all-reduce and all-gather, whose costs can dominate or constrain multi-GPU scaling under some conditions [13]. A training study must therefore account for communication, precision, learning dynamics, and time-to-quality.

Autoregressive LLM inference has a distinct workload shape. Prefill processes prompt tokens, while decoding repeatedly generates new tokens using growing key-value caches. Request concurrency, prompt lengths, output lengths, cache policies, and batching alter how the GPU is exercised. PagedAttention, introduced with vLLM, demonstrated how cache-management design can improve serving throughput: its authors reported a 2–4-times improvement over the systems tested in their 2023 study at comparable latency [17]. That is a historically bounded result, **not** a claim that today’s vLLM is always faster by that factor.

For the flagship study, inference offers the more tractable first domain because kernel shape distributions and time-to-first-token, token latency, throughput, and memory behavior can be instrumented. Training remains a useful extension, not an outcome that may be inferred from inference experiments. A paper that covers both settings should keep their metrics and validity claims separate.

### **3. Related research: contributions that the proposal must build on**

The literature has already moved well beyond toy demonstrations. KernelBench introduced an executable way to measure whether language models can generate correct and faster kernels for selected PyTorch workloads, using a threshold-based success metric [6]. The central insight—that code generation must be tested through execution rather than textual plausibility—underpins the present proposal.

KernelBench-X expands evaluation into 176 tasks and 15 categories and reports substantial variation in semantic correctness and hardware efficiency across task types [7]. Its reported findings include correct kernels that are nevertheless slower than the PyTorch eager baseline. This cautions against treating compilation or

reference agreement as synonymous with optimization. The reported percentages are findings of KernelBench-X's own model and task selection, not general properties of every coding agent.

SOL-ExecBench tests kernels against hardware-grounded speed-of-light targets instead of relying exclusively on mutable software baselines [8]. It also documents benchmark safeguards such as isolation and attention to reward-hacking strategies. This responds to an important comparison problem: beating a particularly slow reference is not the same as approaching efficient execution.

Atrex-Bench explicitly addresses production relevance by sampling operators and shapes from deployment traces, weighting them by observed resource use, and supplying per-problem performance ceilings [9]. The authors also discuss the risk of apparent correctness achieved by delegating back to a framework rather than implementing the target operation. The present proposal **does not claim to invent trace-derived shapes, importance weighting, or hardware ceilings**; those are important prior contributions.

FastKernels explores the translation of generated kernels into representative production-oriented model architectures and inference pipelines [10]. Its project includes multiple levels of integration, not just isolated operators. FlashInfer-Bench likewise provides a trace and implementation schema and mechanisms for testing and substituting kernels in inference engines [12]. These systems reduce the gap between a benchmark runner and deployment, but cannot make results invariant to all hardware, future traffic, precision requirements, or load conditions.

A September 2026 study asking how much real model time generated kernels can affect profiles several workload types, documents limited addressable fractions in some systems, and identifies correctness-test weaknesses that permitted degenerate outputs on certain tasks [11]. Its significance here is methodological: benchmark acceptance conditions and application-level addressability need to be examined together. PTXBench additionally evaluates whether LLM-generated kernels use target architecture-specific instructions on H100 and B200 devices and finds that instruction use alone does not guarantee competitiveness [18].

| Existing research                       | Established contribution                                                                  | Follow-on question emphasized here                                                |
|-----------------------------------------|-------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| KernelBench / KernelBench-X [6–7]       | Executable correctness and efficiency studies across operator suites                      | Does a passing kernel remain valid on held-out shape and precision distributions? |
| SOL-ExecBench [8]                       | Hardware-grounded efficiency targets and robust scoring practices                         | Are efficiency and relative rankings stable after controlled workload shifts?     |
| Atrex-Bench [9]                         | Trace-derived shape selection and importance-weighted scoring                             | How should held-out distribution shift and temporal changes be tested explicitly? |
| FastKernels / FlashInfer-Bench [10, 12] | Inference integration, substitution, and production-oriented workloads                    | How reliably does operator-level improvement transfer to whole-request outcomes?  |
| Real-workload study / PTXBench [11, 18] | Addressable runtime fractions, correctness pitfalls, and architecture-specific evaluation | Which failures are caught by negative controls and cross-hardware validation?     |

**Research gap, stated cautiously.** The literature contains several pieces of the solution. WSTP proposes their combination as a *predeclared shift-and-transfer study*: separate visible and held-out workloads, inspect kernel provenance and numerical edge cases, measure timing with correct asynchronous semantics, and quantify the link to deployment outcomes. Novelty would have to be demonstrated through an implemented comparison with these systems, not merely asserted because the protocol has a new acronym.

## 4. Research questions and falsifiable hypotheses

The experimental unit is a candidate kernel written by a particular generation system for a precisely specified operator. A workload instance specifies shape, layout, dtype, hardware/software configuration, and, where applicable, serving context. An evaluation distribution specifies how often each instance occurs. The *treatment* is the move from the visible development distribution to preregistered held-out and shifted distributions; it is not an undocumented switch to an easier baseline.

**RQ1 — Ranking stability.** When candidate kernels are ordered by correctness-gated speedup on familiar shapes, how often do pairwise rankings reverse under held-out inference-inspired shape distributions?

**H1 (preliminary threshold).** In at least one preregistered operator family with sufficient valid candidates, the pairwise reversal fraction between visible and shifted workload evaluations will be at least

0.20. This threshold is a design choice, not a reported empirical effect. A study should report the full reversal estimate and interval even if H1 is unsupported.

**RQ2 — Correctness transfer.** Which numerical, shape, stride, aliasing, and concurrency conditions reveal failures missed by visible examples?

**H2.** The held-out test battery will detect at least one additional distinct failure class in one or more operator families beyond the visible battery. A zero-failure outcome is a valid and informative result; it must not be disguised as an unsuccessful experiment.

**RQ3 — System transfer.** How well do isolated-kernel improvements predict changes in end-to-end inference latency and throughput under controlled load?

**H3.** For at least one integrated operator, the median application-level improvement will be materially smaller than the corresponding isolated-kernel improvement; the study will preregister the materiality threshold and report the relevant addressable runtime fraction. This is plausible from Amdahl's law but must be measured for the actual application.

**RQ4 — Implementation and architecture dependence.** Do CUDA C++ and Triton candidates differ in dominant failure types or in sensitivity to GPU generation when task difficulty, agent budget, and evaluation contracts are controlled?

**H4.** Failure-category distributions will differ in at least one prespecified operator family. The study should not presume which language wins. Differences might reflect compiler maturity, task design, agent training exposure, or library delegation rather than an intrinsic property of either language.

The hypotheses are intentionally modest and falsifiable. The primary paper should report effect sizes and uncertainty, not reduce the experiment to four binary passes. A confirmatory study must fix family definitions, workload split, multiplicity adjustments, stopping rules, and thresholds **before** inspecting held-out results.

## 5. The Workload-Shift and Transfer Protocol (WSTP)

### 5.1 Phase 0: specification and environment freeze

Each target operator begins with an executable reference and a semantic specification. The specification enumerates legal tensor rank, shape ranges, strides, broadcasting, alignment, dtype, accumulation expectations, mutation/aliasing rules, permitted temporary memory, and output requirements. For an attention operator, it must also define masks, KV-cache layout, positional handling, and any supported quantization. Undefined behavior in the reference must be resolved before comparing candidates.

The environment manifest records GPU model, compute capability, device count, available VRAM, memory mode (including MIG/MPS if used), driver, CUDA toolkit, compiler, relevant libraries, framework versions, container digest, operating system, device clocks when available, and power measurement method. NVIDIA's preview restrictions must be obeyed [15]. The authors should archive command lines, inputs or generators, reference implementation hashes, and result schemas. Public artifacts can omit protected cloud identifiers and confidential traffic while preserving reproducible synthetic or approved workloads.

A practical initial scope is **four operator families**: RMS normalization, general matrix multiplication, reduction or softmax, and attention-associated operations. Include operations with different arithmetic intensity and memory behavior. The exact number of tasks, agent attempts, and repetitions is a planning parameter determined by the hardware budget; it is not legitimate to report statistical confidence from a small convenience sample as though it represented a production fleet.

### 5.2 Phase 1: visible development distribution

A candidate generator receives the operator specification, an API contract, hardware disclosure appropriate to the experiment, and a fixed development budget. It may see the *development* shapes and correctness feedback. It may not receive held-out workloads, private verification answers, or hidden scoring weights. If an agent is allowed to inspect Nsight reports or iterate after failures, every competing condition should receive the same tool access and wall-clock or token budget.

Candidate provenance must be preserved. Save the prompt or instructions (with sensitive data removed), generated source, compiler output, dependency manifest, iteration history, and final binary hash. Each candidate should be associated with its agent/model version and seed where applicable. Models may be stochastic: repeated independent trials, not just the best cherry-picked candidate, are necessary for model-level conclusions.

### 5.3 Phase 2: independently defined held-out workload shapes

The study uses three related evaluation sets. **Set A, visible/static**, reproduces published or documented development shapes. **Set B, held-out/interpolative**, samples unexposed combinations inside the declared legal range. **Set C, shifted/stress**, emphasizes long sequences, non-power-of-two dimensions, low and high batch sizes, noncontiguous strides where legal, different alignment, and precision or cache states actually supported by the operator. Held-out inputs must be generated independently of candidate development and retained as a fixed evaluation release.

Distributions can be derived from public workload traces or fully disclosed synthetic generators. They must be labeled accurately: a synthetic long-tail shape distribution is *inference-inspired*, not proof of production representativeness. Where licenses permit, a study can compare a public trace-derived distribution with an equally sized synthetic distribution to estimate the consequences of sampling choice. Shape-provenance metadata should accompany every recorded score.

For example, a preregistered attention-related suite might include short and long contexts, power-of-two and irregular head dimensions where supported, different contiguous and strided layouts, and a range of cache occupancy. This is **illustrative design**, not a claim that a particular GPU, kernel, or model handles every combination. Unsupported points remain explicitly excluded and counted as such, rather than silently dropped after a failure.

### 5.4 Phase 3: correctness, integrity, and negative controls

Correctness should be a gate that is checked before performance is awarded. A stronger battery has several layers:

1. **Execution validity:** compilation, launch, completion, output shape, legal memory access, and no hidden asynchronous error.

2. **Semantic agreement:** values, mutation behavior, strides, boundary handling, and supported aliasing against one or more reliable references.
3. **Numerical validity:** absolute, relative, and scale-aware error; NaN/infinity behavior; reduction sensitivity; and documented tolerance tied to format and application.
4. **Robustness:** held-out dimensions, layouts, repeated runs, permitted concurrency, and precision settings.
5. **Task-level validity:** where integrated, a downstream model-quality or service-output check, clearly distinguished from bitwise reproducibility.

A generic condition `abs(candidate - reference) <= atol + rtol × abs(reference)` can be useful, but tolerances must be specified with numerical expertise rather than enlarged until every kernel passes. Additional checks should be designed for near-zero outputs, constant or zero-producing candidates, unwritten output regions, and unexpectedly cheap operations. The 2026 real-workload study provides a concrete warning that a permissive closeness check can accept degenerate outputs in some benchmark cases [11].

**Negative controls** are intentional invalid or uncompetitive candidates used to test the evaluator itself: a kernel that writes zeros, a partial-output kernel, a shape-specialized kernel, an asynchronously failing kernel, a no-op, or a candidate that delegates to an explicitly prohibited fallback. The evaluator should reject such cases for the correct reason. This is an evaluation-integrity test, not a claim that all fallback use is illegitimate: vendor libraries are proper baselines, while fallback restrictions depend on the defined kernel-generation task. Acceptable delegation must be documented in advance.

The scorer must not expose hidden expected outputs, ground-truth fixture files, or derived acceptance thresholds to candidate code. Candidate execution should have controlled file-system and network access and should not be able to modify the reference, evaluator, timing scripts, or results. A successful isolation check is a property to verify, not a property to assume because a workload happens to run inside a container.

## 5.5 Phase 4: asynchronous-safe performance measurement

CUDA launches are commonly asynchronous with respect to the host. A host timer wrapped around a launch can measure submission rather than completed device work. The official programming guide documents CUDA event timing and synchronization as mechanisms for accurate stream-level measurement [1]. The protocol should perform a correctness pass, establish a consistent warmup strategy, and use CUDA events in the relevant stream or synchronized end-to-end wall-clock timing for the system metric.

Timing policy must define whether compilation, autotuning, graph instantiation, allocation, and data transfer belong inside the measured interval. No one policy is universally correct. **Steady-state kernel latency** may exclude one-time compilation and allocation, while **cold-start application latency** must include relevant setup. A publishable report includes both where the distinction affects decisions. CUDA Graph executions are compared only under equivalent legal capture and replay conditions; the graph itself must not silently exclude work included in another candidate [1].

Within each workload cell, collect enough independent measurement groups to estimate uncertainty. A reasonable starting design is at least 30 timed groups per configuration, after warmup, with implementation order randomized or interleaved. The actual sample size should be justified with pilot variance and the smallest practically meaningful effect. Report medians and interval estimates; preserve raw run-level records, not just a single best time. GPU clock behavior, temperature, contention, cache state, and background workloads must be controlled or monitored. Do not profile and time the same run unless the profiling overhead has been explicitly addressed.

The reported per-cell speedup can be defined as  $\text{median}(\text{reference latency}) / \text{median}(\text{candidate latency})$ , for *correct* candidates. The primary aggregate should preserve the workload's predeclared weighting. A weighted mean of log-speedups is one interpretable choice:  $L = \sum_i (\text{weight}_i \times \log(\text{speedup}_i))$ , with weights nonnegative and summing to one. Then  $\exp(L)$  is the weighted geometric-mean speedup. Invalid cells must be **reported separately with a coverage rate and failure type**; an aggregate

over only the successful cells risks concealing systematic failures. Where a single operational decision score is needed, prespecify an invalid-candidate penalty and publish the components alongside it.

For diagnostic explanations, Nsight Systems can identify CPU gaps, stream overlap, and graph behavior; Nsight Compute can measure instruction, memory, occupancy, and roofline-related data [4–5]. However, profiler-derived explanations should be presented as mechanisms consistent with the observations, not assumed causal proof unless an intervention isolates the mechanism.

## 5.6 Phase 5: hardware and deployment transfer

For cross-hardware inference, rerun supported kernels on a second disclosed architecture if access permits; for example, a Hopper-class device and a supported Blackwell-class device. Do not infer portability from compiler success alone. Some operators and data formats may not be comparable across devices; such cells belong in a support matrix. The comparison should distinguish a **portable source implementation** from an **architecture-tuned implementation**, and both from a runtime library that selects different kernels per device.

To study end-to-end transfer, integrate only candidates that pass held-out correctness tests into a small, controlled inference application. Use the same model weights, request distribution, scheduler, batching policy, cache configuration, and precision as the baseline. Record time to first token, inter-token latency, aggregate throughput, p50/p95/p99 request latency, peak and steady-state VRAM, GPU utilization, and correctness or output-quality checks. Prefill and decode must be measured separately where relevant. The experimental report should disclose offered load and concurrency; throughput at saturation and latency at low load answer different questions.

The **addressable runtime fraction** establishes an interpretable ceiling. If an operator consumes fraction  $f$  of baseline runtime and the replacement is  $s$  times faster, the simple Amdahl estimate is  $\text{overall speedup} \leq 1 / ((1 - f) + f/s)$  under fixed workload assumptions. Queueing, memory pressure, batch size, and overlap may cause actual results to depart from this simplified bound. Therefore, the Amdahl estimate is a diagnostic baseline, not a substitute for integrated measurement.

6. Metrics, statistical analysis, and decision rules

A careful report separates what is measured from what is inferred. The primary outcome for RQ1 is the *pairwise ranking reversal fraction* across valid candidate pairs evaluated on both distributions. Let  $R(A,B)$  be the fraction of candidate pairs whose ordering reverses between distributions A and B. Candidate pairs that are invalid on either distribution should be reported in a separate survival/coverage analysis rather than quietly excluded from the study’s headline result.

A secondary outcome is rank correlation (Spearman’s rho or Kendall’s tau), with bootstrap confidence intervals computed at an appropriate level. When observations share kernels, tasks, or random seeds, simple per-cell resampling understates dependence. A hierarchical bootstrap over tasks and agent runs, or a mixed-effects model with disclosed assumptions, is more defensible. Multiple hypotheses and operator-family subanalyses require a specified adjustment or an explicit distinction between confirmatory and exploratory findings.

| Research dimension | Primary reported evidence                                                    | Why a single speedup is insufficient                        |
|--------------------|------------------------------------------------------------------------------|-------------------------------------------------------------|
| Correctness        | Pass fraction and taxonomy over all legal cells; output error statistics     | Some shapes or numerical regimes may fail while others pass |
| Ranking stability  | Reversal fraction and rank correlation across A/B/C                          | An apparent winner can depend on the test distribution      |
| Kernel efficiency  | Latency distribution, memory traffic, hardware-bound comparison              | A slow reference can exaggerate the importance of a win     |
| System transfer    | TTFT, inter-token latency, throughput, p95/p99, runtime fraction             | Local kernel improvements may be small in an application    |
| Portability        | Support matrix and cross-device relative performance                         | Different architectures support different paths             |
| Cost and energy    | Measured energy when available, run-rate assumptions, cost per useful output | Raw speed alone does not imply economic efficiency          |
| Research integrity | Reproducible source, version manifest, negative-control outcomes             | A benchmark can reward shortcuts or non-repeatable results  |

For H1, the paper should report the reversal fraction and its uncertainty even when the threshold is not met. For H2, failures should be grouped into distinct root-cause classes with examples and verified reproductions; multiple manifestations of one bug should not be counted as independent failures. For H3, compare

measured end-to-end improvement with the Amdahl estimate and identify scheduling or memory effects that might explain any difference. For H4, use comparisons that match tasks and generation budgets; descriptive differences without power or an appropriate model are not evidence of a universal language advantage.

**Minimum acceptance rule for a deployment candidate.** A candidate passes only when it (a) satisfies the agreed semantic and numerical contract over the relevant workload support set, (b) improves at least one prespecified service-level metric with uncertainty that excludes an unacceptable regression, (c) respects memory, safety, and operational budgets, and (d) can be reproduced from a pinned artifact. These are proposed criteria to tune with stakeholder requirements; no universal threshold is claimed.

## 7. GPU infrastructure and the economics of adoption

NVIDIA GPU selection is not a function of peak theoretical FLOP/s alone. For large-model training, memory capacity, memory bandwidth, interconnect topology, collective communication, fault tolerance, and time-to-quality matter. For inference, the important parameters also include request mix, model size and quantization, prefill/decode balance, scheduler behavior, and tail latency. An expensive GPU may offer strong throughput at high occupancy yet yield poor unit economics for short, sporadic tasks. Conversely, a cheaper device may force memory offloading or smaller batches that erase its apparent price advantage.

A research artifact should make these trade-offs auditable without pretending to know every organization's internal cloud contracts. One useful financial metric is  $\text{cost per 1,000,000 useful output tokens} = \text{total attributable serving cost} / \text{useful generated tokens} \times 1,000,000$ . The numerator must disclose which costs are counted: instance time, idle provisioning, orchestration, storage, network, and any development or compilation amortization. Provider, region, purchase terms, currency, date, and utilization assumptions should accompany comparisons. Model quality, request failures, and service-level obligations must be held within stated bounds.

Energy can be estimated from time-resolved telemetry where the hardware and permissions allow it. However, board power samples, GPU package power, and whole-system electricity are not equivalent. If the study measures only device telemetry, it must say **GPU-reported energy**, not data-center electricity or full life-cycle

carbon footprint. Energy per accepted request or per useful token is often more meaningful than instantaneous watts, provided the workload and quality gate are specified.

A feasible adoption path is incremental. First benchmark a small number of targeted kernels and record diagnostic profiles; then integrate validated improvements behind an explicit switch; finally run representative request loads with a rollback path. Avoid assuming that a generated kernel should replace a mature vendor primitive simply because it is newer. CUTLASS, cuBLAS-backed paths, and other vendor implementations provide critical baselines; replacing them is a high bar, not a default goal [2, 11].

## 8. Worked illustrative case: normalization under shape shift

Consider a *hypothetical* RMS normalization kernel for a decoder-only model. The reference defines the reduction axis, epsilon value, accumulation precision, output dtype, and permitted tensor layouts. A generation agent sees a small visible set: batch sizes 1 and 8 with convenient hidden dimensions and contiguous memory. An optimized candidate fuses intermediate operations and appears faster in microbenchmarks.

WSTP withholds a larger test grid until the final submission. It includes batch size 2, non-power-of-two hidden dimensions supported by the operator, odd alignment boundaries, and legal strided inputs. A numerical test measures error near zero and at extreme but in-range magnitudes. The candidate is then checked for complete writes and memory safety, not only averaged output closeness. If it fails one legal held-out case, it cannot be advertised as universally correct for the full contract. If it passes all cases but loses speed at small batches, the paper reports that limitation rather than hiding it inside a favorable average.

Next, the researchers identify the fraction of serving time attributable to the target normalization calls. A replacement that is faster for those calls may still produce negligible overall improvement if attention and GEMM dominate. The candidate is integrated into a pinned model with a fixed request trace, and p50 and p95 request latencies are measured at several loads. The final result could favor the candidate, show no meaningful benefit, or reject it. **No particular result is assumed in this example.**

The illustration highlights why the proposal has value even when generated kernels do not outperform hand-tuned code. A defensible negative result can reveal missing shape coverage, a misleading baseline, or a flawed timer. Such failures can become labeled examples for future agent training and evaluation, provided the data sources and redistribution rights permit publication.

## 9. Reproducibility, responsible evaluation, and threats to validity

**Construct validity.** The terms *correct*, *fast*, *portable*, and *deployment-ready* depend on a declared workload and service objective. Numerical tolerance that ignores scale or the application's semantics can misclassify wrong results. A hardware roofline number cannot certify application usefulness. A synthetic distribution can model an interesting stress case without representing customer traffic.

**Internal validity.** Asynchronous launches, incomplete warmup, compilation effects, stale cache state, allocator variation, overlapping workloads, clock changes, and instrumentation overhead can distort measurements. Interleaving runs and recording temperature/clock state reduce some bias but do not prove its complete absence. Negative controls should be run against the actual final verifier, not a simplified mock.

**External validity.** A study on one GPU architecture, one driver, a small model, or one scheduler does not establish fleet-wide generality. Heterogeneous GPUs, MIG/MPS configurations, different memory sizes, and cloud contention require independent trials. CUDA 13.4's Rubin developer preview is explicitly outside production benchmarking claims [15].

**Statistical validity.** Repeated timings of the same candidate on one GPU are not independent samples of agent ability or architecture support. The study should distinguish kernel-level repetitions, different task instances, independent model generations, and genuinely independent hardware or workload conditions. Favor paired comparisons when possible and avoid selecting only the best of many attempts without disclosing the selection budget.

**Evaluation integrity.** Hidden holdout tests and scoring scripts should be inaccessible to generated code. Candidate processes must not read evaluator-owned answers, mutate baselines, or use runtime

APIs outside the agreed contract. A scorer should reject an implementation that delegates forbidden work back to the host or hides computation outside the timed region. Tests of these defenses should be performed with purpose-built negative controls. This does not mean forbidding ordinary library use in all experiments; baseline rules must match the task’s scientific purpose.

**Research ethics and reproducibility.** Production logs may contain sensitive prompts, request metadata, or model details. Public artifacts should use approved traces, aggregated workload metadata, or disclosed synthetic generators. Research teams should respect third-party repository licenses and avoid claiming authorization from NVIDIA or model providers. The best defensible practice is to publish a reproducibility capsule: task specifications, hashes, workload generators, dependency locks, timing scripts, scoring rules, anonymized measurements, and the exact analysis notebook.

The limitation of this paper is clear: it proposes, rather than validates, WSTP. Whether the protocol produces new insights beyond Atrex-Bench, FastKernels, FlashInfer-Bench, or SOL-ExecBench cannot be settled from literature synthesis alone. An implementation should begin by reproducing at least one baseline result or failure mode from prior work, then run the preregistered shift-and-transfer study. Unanticipated findings must be labeled exploratory rather than rewritten as if they were predicted.

## 10. A practical research agenda and implementation roadmap

An efficient study proceeds through milestones with explicit stopping conditions. The first milestone is a specification and benchmark-integrity audit: freeze a small supported operator family, port a trustworthy reference, write meaningful negative controls, and verify that CUDA event timing agrees with an independent synchronized path within a documented tolerance. Do not introduce agent-generated code before the verifier can reject known-wrong kernels.

The second milestone is a kernel baseline and profiling suite. Establish PyTorch/library references, one well-understood hand-written implementation when feasible, and a Triton implementation. Document supported shapes, precisions, and hardware. Compare the actual compiled paths, not language labels. Establish the device

performance envelope and use Nsight to identify mechanism-level differences. The deliverable is a reproducible baseline, not an impressive headline speedup.

The third milestone is controlled agent generation. Run one or more named model configurations across the same tasks with fixed budgets, then preserve all attempts rather than publishing only the best. Produce source hashes, build logs, generated-code provenance, and scores on the visible set. The fourth milestone is the preregistered held-out test: freeze candidate submissions; evaluate Set B and Set C; estimate correctness and ranking stability; and audit negative-control outcomes.

The fifth milestone is deployment transfer. Insert a subset of fully correct candidates into a small inference model and measure service-level outcomes. If the targeted kernels account for negligible runtime, report that negative finding and prioritize a more addressable operator rather than moving the goalposts silently. A second GPU generation, if available, increases the scope of portability claims but is not a substitute for full version disclosure.

Finally, release the study as an artifact-backed research publication. A compelling academic manuscript would include **actual** result tables, confidence intervals, profiler evidence, ablation tests, failure-case reproductions, and a comparison with at least one existing kernel benchmark. Until these measurements exist, the appropriate output remains a rigorously defined research perspective and protocol, not an empirical conference paper claiming novel performance gains.

## 11. Relevance to Deaimer and to GPU research teams

Deaimer publicly emphasizes realistic, reproducible, objective evaluation of AI agents and the difference between plausible answers and completed technical work [19]. GPU kernel generation is a natural case study for this principle because the outcome can be tested at several levels: source compilation, numerical agreement, memory safety, hardware efficiency, and end-to-end service behavior. That connection is **methodological**. This paper does not claim that Deaimer owns an NVIDIA GPU benchmark product, operates a GPU research laboratory, or has an NVIDIA partnership.

A future Deaimer research program could investigate how to construct trustworthy executable environments for GPU-coding agents, how to prevent reward shortcuts, and how to collect consented and licensed failure cases for evaluation or training. The results would be meaningful to organizations developing models that write CUDA/Triton and to infrastructure teams weighing automated optimizations. Whether such work becomes an offering depends on implementation, internal validation, available hardware, and organizational decisions; it should not be presented as a currently shipped capability.

For NVIDIA and other GPU software teams, the relevant contribution is a shared evidence standard, not promotional language: reproducible measurements, compiler and device disclosure, numerical acceptance criteria, and clear separation of promising kernel-level results from demonstrated application gains. For an individual researcher, the same work can become a strong professional portfolio when paired with a public, independently runnable implementation and transparent negative results. The article alone demonstrates synthesis and experimental design; **measured results and a reviewed artifact** are what would support stronger claims of original empirical research.

## 12. Conclusion

CUDA performance research in 2026 spans low-level execution, tile-based languages, mixed-precision matrix engines, graph scheduling, multi-GPU communication, and serving infrastructure. AI-generated CUDA and Triton kernels add an important new challenge: the ease of producing a candidate is not equivalent to evidence that the candidate is correct, efficient, and deployable. Contemporary benchmarks already provide substantial advances in correctness testing, hardware-grounded efficiency, trace-derived workloads, and production-oriented integration. The next useful research step is to study *transfer*: from visible to held-out shapes, from a numerical check to a full semantic contract, from one GPU generation to another, and from a microbenchmark to service-level outcomes.

The proposed Workload-Shift and Transfer Protocol makes those transitions explicit and testable. Its essential discipline is to predefine the workload distribution, correctness and timing contract, baselines, uncertainty analysis, and acceptance rules, while preserving failures as evidence rather than filtering them away. The

proposal is suitable for implementation by an academic or industry team and for subsequent peer review. The decisive conclusion must come from reproducible experiments, not from the promise of faster kernels alone.

## References

- [1] NVIDIA. **CUDA Programming Guide**, sections on asynchronous execution, streams, and CUDA Graphs. <https://docs.nvidia.com/cuda/cuda-programming-guide/>
- [2] NVIDIA. **CUTLASS Documentation**, overview and architecture-dependent mixed-precision support, version 4.8 (September 2026). <https://docs.nvidia.com/cutlass/latest/overview.html>
- [3] OpenAI Triton project. **Triton Programming Guide**. <https://triton-lang.org/main/programming-guide/chapter-1/introduction.html>
- [4] NVIDIA. **Nsight Systems User Guide** (CUDA trace and graph profiling). <https://docs.nvidia.com/nsight-systems/UserGuide/index.html>
- [5] NVIDIA. **Nsight Compute Profiling Guide** (roofline and kernel metrics). <https://docs.nvidia.com/nsight-compute/ProfilingGuide/index.html>
- [6] Ouyang, A., et al. (2025). **KernelBench: Can LLMs Write Efficient GPU Kernels?** ICML, Proceedings of Machine Learning Research 267. <https://proceedings.mlr.press/v267/ouyang25a.html>
- [7] Wang, H., et al. (2026). **KernelBench-X: A Comprehensive Benchmark for Evaluating LLM-Generated GPU Kernels**. arXiv preprint 2605.04956. <https://arxiv.org/abs/2605.04956>
- [8] Lin, E., et al. (2026). **SOL-ExecBench: Speed-of-Light Benchmarking for Real-World GPU Kernels Against Hardware Limits**. arXiv preprint 2603.19173. <https://arxiv.org/abs/2603.19173>
- [9] Yang, L., et al. (2026). **Are LLM-Generated GPU Kernels Production-Ready? A Trace-Driven Benchmark and Optimization Agent**. arXiv preprint 2607.14541 (Atrex-Bench). <https://arxiv.org/abs/2607.14541>

- [10] Oliaro, G., et al. (2026). **FastKernels: Benchmarking GPU Kernel Generation in Production.** arXiv preprint 2605.23215. <https://arxiv.org/abs/2605.23215>
- [11] Agarwal, G., Garg, A., and Singhal, I. (2026). **How Much of a Real Workload Can LLM-Generated GPU Kernels Actually Reach?** arXiv preprint 2609.21058. <https://arxiv.org/abs/2609.21058>
- [12] Xing, S., et al. (2026). **FlashInfer-Bench: Building the Virtuous Cycle for AI-driven LLM Systems.** arXiv preprint 2601.00227. <https://arxiv.org/abs/2601.00227>
- [13] NVIDIA. **NCCL 2.32.3 User Guide: Overview of NCCL.** <https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/overview.html>
- [14] NVIDIA. **CUDA Toolkit Archive** (listing CUDA Toolkit 13.4.3, October 2026). <https://developer.nvidia.com/cuda-toolkit-archive/>
- [15] NVIDIA. **CUDA Toolkit 13.4 Release Notes** (including Vera Rubin developer-preview restrictions). <https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html>
- [16] NVIDIA. **cuTile Python Release Notes**, version 1.6.0 (September 9, 2026). [https://docs.nvidia.com/cuda/cutile-python/generated/release\\_notes.html](https://docs.nvidia.com/cuda/cutile-python/generated/release_notes.html)
- [17] Kwon, W., et al. (2023). **Efficient Memory Management for Large Language Model Serving with PagedAttention.** ACM SOSP; arXiv 2309.06180. <https://arxiv.org/abs/2309.06180>
- [18] Zhang, G., et al. (2026). **PTXBench: Benchmark and Adapt LLMs for GPU Kernel Optimization with Architecture-specific PTX.** arXiv preprint 2608.17379. <https://arxiv.org/abs/2608.17379>
- [19] Deaimer. **AI Agent Benchmarks and Evaluation Research** (public benchmark principles), accessed October 19, 2026. <https://www.deaimer.com/benchmarks>

*Source-status note:* The 2026 arXiv works cited above are identified as preprints unless another formal venue is explicitly stated. Historical benchmark results are attributed to their original papers. Software documentation may change after the date of this article. No part of the proposed WSTP has been experimentally validated for this publication.